Super High Compression of Line Drawing Data

by

David B. Cooper
Division of Engineering
Brown University

April, 1976

Work supported by NASA Grant NSG 5036.

1.0 Introduction

In this report we describe models which can be used for accurately representing the type of line drawings which occur in teleconferencing and transmission for remote classrooms and which permit considerable data compression, thereby reducing the cost of communicating such pictures. (This methodology would, of course, be equally effective in reducing line-drawing picture storage costs.) In conferencing, one is usually dealing with drawings which can be made on a pad, with draftsman prepared charts or with slide or viewgraph projections. In the remote classroom application, the pictures are drawn on the blackboard. Our interest is in encoding these pictures in binary sequences of shortest length but such that the pictures can be reconstructed from these sequences without loss of important structure. Taking our examples from typical blackboard pictures arising in lectures on elementary electronics and on histology, e.g., Figures 1-5, we show that exploitation of reasonably simple structure permits compressions in the vicinity of 100 to 1 for the former and 30 to 1 for the latter and we indicate how additional compression might be possible at the cost of considerably more information processing.

Compression is achieved in this approach through a certain level of picture understanding and through nonlinear approximation. For the most part, when dealing with highly stylized material such as electronic or logic circuit schematics as appear in a lecture on electronics, it is unnecessary to reproduce configurations exactly. Rather, the symbols and configurations must be understood and be reproduced, but one can use fixed font symbols for resistors, diodes, capacitors, etc. Lines should be sharp and smooth, but curved lines representing wires can be straightened and sizes and relative positions need not be accurately reproduced. Some measure of picture understanding is required for this. Compression of pictures of natural phenomena such as the tissue drawings in figures 3-5 can be realized by taking a similar approach, or essentially zero error reproducibility can be achieved but at a lower level of compression. Both of these approaches are based on the use of primitive pictures and operators described by a few parameters for approximating subpictures. For example, circles, elliptic arcs, splines using well separated knots, etc. permit approximations to the pictures of interest. The many small circles in figure 5 or the Purkinje cells in figure 4b can be approximated by replication of a basic circle or basic almost-solid black oval, respectively. In one approach, the regular curve approximations would be randomly perturbed to appear more natural--though with no improvement in fit. In another approach, the regular curve approximations would be perturbed

to more accurately represent the original data.

Is the pattern recognition required to realize this encoding practical? If the communicator exercise some care in drawing, it should be possible to develop recognition algorithms which would be very practical. Some Computer graphics terminals now have a capability of recognizing hand drawn alphanumeric symbols and there are commercial optical character readers which can handle these symbols. Furthermore, if subpictures are encountered which are not recognizable, the processor could use simpler compression techniques which achieve less compression. At the end of the report we have included a section with a few simple experimental results and some discussion in support of our contention that the recognition problem is manageable.

The coding described in the report is directed primarily at redundancy removal and little consideration has been given to the most useful encoding for other purposes. However, the modelling of picture structure is basic to a number of other very interesting and important problems. Among these are the following.

(1) As already mentioned, efficient document and more generally line drawing date storage.

(2) Picture recognition by computer. There is growing interest now in scene understanding by computer for a variety of applications. A use for sophisticated computer understanding of line drawings would be to provide an increased capability for human communication with machines.

(3) Nonlinear approximation theory.

(4) Tradeoffs in man-machine interaction and cooperation. A variety of things could be explored here. The machine could assess recognition and encoding costs for various portions of a picture and choose encoding methods optimizing some measure of computation and communication costs. Another interesting area of exploration is the saving in pattern recognition cost achievable through the communicator providing certain recognition aide to the machine. For example, were the drawer to label a subpicture such as full wave rectifier or grounded emitter amplifier, etc., with its descriptive name, the label could be easily recognized thereby considerably simplifying the circuit schematic recognition necessary for encoding the subpicture to achieve the greatest levels of compression. (Labelling of tissue pictures in the histology course is almost always done.)

The development of coding methodology for the pictures of interest here is an evolutionary process. The codes to be described are not necessarily in their final form. Basically, we use primitives which are symbols and simple geometric patterns which can be described by a few parameters, and more general structures which we call groups. The latter are a combination of lists and graphs. Some groups are for general application and handle point sequences, curve segment sequences, region description, and subpattern replication. These result in encoding in which groups are encoded in terms of groups. Other groups are tailored to specific subject material such as electronic circuit schematics. These may call upon the former or may simply contain as an integral part one or more structures which are similar to some of the former types. Refinement and further development of the procedures we have proposed will require testing through experimentation.

There is overlap in philosophy in some approaches being taken in pattern recognition, artificial intelligence, and computer graphics. Our approach shares some similarities with aspects of Sketch Pad [1] and is in harmony with some of the current thinking in knowledge understanding systems. However the emphasis in ours is on natural pattern approximation, a modest level of picture understanding, and pattern recognition. The motivation for our approach is the mathematical theory for source coding for achieving data compression. Though we do not make much explicit use of this theory, our procedures are greatly influenced by such concepts as Huffman coding, predictive coding, universal coding [2], and more generally by the concept of representing pictures as elements in relatively small hierarchical universes such that picture elements of different universes at the same level are relatively statistically independent.

Introductory background material on the approach taken in this report is contained in an earlier paper [3] appended to the end of the report.

2.0 Electronic Circuits

In this section, we present those structures which are obviously effective in compressing the material appearing on a blackboard in an introductory electronics course. The structures we deal with will realize compressions of roughly 100 to 1 at what appears to be a reasonable and practical level of pattern recognition. Reconstructed pictures here would use a fixed symbol font, and circuit schematic elements and curve locations and size would be roughly preserved. This can be achieved at a low understanding of the picture. For example, the required level of understanding here is that a subpicture is a group such as the electronic circuit group, or the logic circuit group, or the transistor collector-current versus collector-to-emitter voltage group, etc., and recognition of the primitives such as circuit component symbols or coordinate axes, etc. Additionally, a circle or rectangle about a portion of a circuit (used to call attention to the portion of the circuit) must be recognized as a closed curve which is not part of the circuit. Considerably greater compression is possible if, e.g., it is unnecessary to fairly well reproduce the spatial configuration of an electronic circuit diagram but is only necessary to produce a circuit diagram with correct components and connections. However, the required pattern recognition is considerably greater in the latter case if it is necessary to reproduce subcircuit labels and English statements which are directed at a portion of the circuit and depend on their proximity to the appropriate subcircuits in order that the statement meanings be clear. We return to this matter at the end of the section.

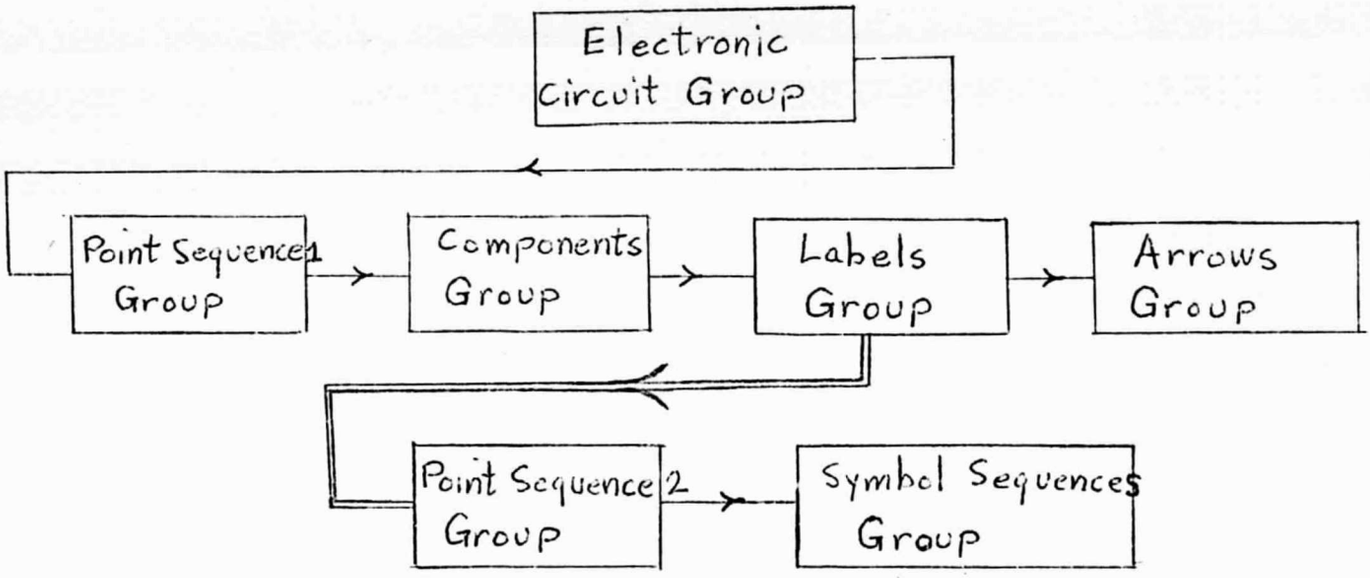
We introduce three groups which would provide for much of the compression achievable with the blackboard pictures appearing in an elementary electronic circuits course. A few additional groups would be helpful, e.g., a Semiconductor Junctions Group for encoding pictures of blocks of semiconductor materials for NP or NPN or PNP junctions and containing bound and unbound charge symbols.

2.1 Electronic Circuit Group

A realization of the electronic circuit group is any circuit schematic composed of wires, resistors, capacitors, inductors, diodes, transistors, transformers, operational amplifiers, A.C. sources, and batteries. (Other components such as the ground symbol, vacuum tubes, etc. can also be included.) Figure 1 is a copy of a portion of the contents of the blackboard during a lecture in an

introductory electronics course. We shall use the schematic of the transistor amplifier there, figure 1a, as the example to illustrate a number of statements in the following paragraphs. By a "node" we will mean a point of intersection of 3 or more wires, or roughly the midpoint of a straight or smoothly curved wire connecting two circuit components, or a terminal of a component such as a transistor, or a point on a wire at which the wire goes through an abrupt large change in direction. In the example shown, there are 13 nodes. They are labelled with circled numbers to facilitate our discussion here.

The Electronic Circuit Group can be considered to consist of four subgroups as shown, and the Label Group in turn can be decomposed into two subgroups.



A discussion of the contents of these groups follows.

In order to resolve a resistor in the figure, at least 3 resolution cells per straight line segment in the resistor symbol are necessary. Then a field of roughly 256 by 256 resolution cells would be adequate for representing the example. (Actually, more than 256 resolution cells are required for the horizontal extent and a finer resolution might be necessary to represent the labels. Hence, the 256 x 256 = 65,536 cell field is a very conservative figure and the compression ratio to be achieved using our proposed methods should be higher than the number we present.)

Point Sequence 1 Group contains the information necessary for locating all circuit nodes. More specifically, the circuit node point sequence is specified such that all pairs of nodes which are terminals of at least one circuit branch appear in the sequence. However, there may also be pairs of successives nodes in the sequence which are not terminal nodes of the same circuit branches. Next, the circuit element appearing in each branch is specified in the Circuit Component Group. A branch may contain a two-terminal circuit element (e.g., a capacitor or a diode) or it may represent a pair of terminals for an element such as a transistor or a transformer having three or more terminals, or it may contain a wire only. Also, in a few instances there will be no branch between a pair of successive nodes. More specific details are the following. The node sequence is given as the changes in coordinates of the locations of the nodes. For circuits, we observe that most branches are horizontal or vertical. Hence, instead of giving x, y pairs for each node in the sequence, we use a 1 or 0 to specify horizontal or vertical branch, respectively, as appropriate. The change in the remaining coordinate in moving from a node in the sequence to its successor is then encoded. Specification of nodes requiring both x and y encoding is provided near the beginning of the code word. The coordinate system to be used here is a local rectangular coordinate system with the extent of the coordinate range in each direction being the sides of the smallest rectangle encompassing the circuit schematic. These values must be encoded near the beginning of the group code word. For the circuit in figure 1a, the node sequence information might appear as follows, where by x_i we mean the binary representation for the change in the x components of the nodes labeled i - 1 and i and similarly for y_i.

$$x_4, y_4, x_{10}, y_{10}, x_4, y_4, 1, x_{13}, 0, y_8, 1, x_7, 0, y_6, 1, x_5, 0, y_4,$$

$$1, x_3, 1, x_2, 0, y_1, 1, x_{12}, 0, y_{10}, 0, y_{12}, 1, x_{11}, 0, y_8$$

Among all permissible node sequences under consideration this one contains the minimum number of node coordinate appearances. Also, most of the coordinate changes to be encoded are roughly of the same size! The exceptions such as the change in the x coordinate in going from node 1 to node 12 appear only once! Hence, universal coding can provide some additional compression here.

Next, information concerning branch content is provided in the Circuit Components Group. Since on occasion, spurious effects such as stray capacitance are represented by circuit elements symbols in broken line form, the number and location of branches to be displayed in broken-line form is given. Finally, branch content is provided with

a sequence of Huffman codewords--one for each branch. A code, which might be improved slightly but which is used for purposes of compression estimation here, is a 1 or 0 signifying wire or component, respectively, followed by a three bit word for each of resistor, capacitor, inductor, A.C. source, battery, diode, transistor, and five bit words for transformer, operational amplifier, or no connection. A transistor codeword is followed by a three bit code word specifying the connections of the base, emitter, and collector to the three nodes given, and a four terminal transformer is followed by a five bit code word specifying the primary and secondary winding terminals associated with the four nodes given. A five terminal transformer--a split primary or a split secondary--would be treated for convenience by assuming that all the primary winding nodes or all the secondary winding nodes were given first and a primary and a secondary node with the same sign sense appeared as successive nodes in the node sequence. Hence, the code word following that for five terminal transformer would consist of a 1 or a 0 indicating whether the primary or the secondary winding nodes appear first and then a 1 or a 0 indicating which of the primary or the secondary winding has three nodes. The operational amplifier input nodes are encoded sequentially in the Point Sequence 1 Group. Hence, the operational amplifier symbol identification codeword in the Components Group is followed by a 1 or 0 indicating that the two input nodes precede or follow, respectively, the output node.

Labels and Arrows

Referring to the example, we see a number of labels and arrows associated with the circuit. For example, there is a label of 4.7k associated with the resistor between nodes 3 and 4, a label, R_L = 1k, associated with the resistor between nodes 13 and 8, a curved arrow in the base-emitter circuit loop, etc. These symbols are integral parts of the circuit schematic and hence are treated as part of the Electronic Circuit Group. We treat each label as a symbol sequence and the set of labels is encoded as a Symbol Sequences Group. An appropriate sequence of symbol sequences here is 47k, I_s, B, E, V_{CE}, R_L = 1K, C, R_B, I_b. The position of the first symbol in each of these symbol sequences is available in the Point Sequence 2 Group The change in coordinates in going from one such symbol to the next are the quantities encoded. For the Symbol Sequences Group, following the group identifier is a code word specifying the common font, and then each symbol sequence is encoded. For each such sequence, we specify the number of symbols in the sequence, then a sequence of code words--one for each

symbol in order to identify the symbol. Since subscripting occurs frequently, each symbol code word (the first symbol excepted) is to be preceded by a 1 or 0 indicating that the symbol is to be at the same height or lowered slightly with respect to the preceding symbol.

Finally, there is the matter of arrows. We encode them as a group separate from the group Symbol Sequence. The arrow is described by a sequence of points through which the arrow shaft passes--these points including the arrow endmost points--and the location of the head or heads. A 1 or a 0 specifies whether there is one head or two heads, respectively, and in the case of the former, this is followed by a 1 or a 0 indicating that the head is attached to the first or to the last point in the point sequence describing the arrow. Some smooth curve algorithm is used for passing arrow shafts through the specified points. For a straight arrow such as the one running from the branch, connecting nodes 12 and 11, to the branch connecting nodes 4 and 13 in the example, only the two endpoints need specifying. For the curved arrow in the base-emitter circuit loop, three points should provide adequate representation. If a label intersects the line of the arrow shaft as is the case in the example where v_{CE} and the line of the first arrow shaft intersect, the reconstruction algorithm is to blank the shaft in the vicinity of the label.

The sequence of code word lengths for this group is as shown.

Point Sequence 1 Group

$n_1$bits---specify the number of node appearances in the node sequence for a circuit. There are 17 node appearances in the node sequence for the example.

$n_2$bits---specify the number and positions in the node sequence of those nodes for which both x and y coordinate changes are given. Assuming many fewer than $\frac{n_1}{2}$ nodes in the sequence will be of this type, the number of bits required here is greatly over bounded by $(n_1/2) \log n_1$ with $n_1/2$ an upper bound on the number of such nodes and $\log n_1$ the number of bits to locate the position in sequence of each such node. There are three such nodes, in the first three positions in the node sequence, for

the example. Hence, about 18 bits are required here.

$n_3$bits---specify the changes in coordinates in successive nodes in the node sequence. Also, the one bit identifiers of horizontal or vertical branches are included here. There are 20 coordinate changes and 14 branch orientation bits for the example. Since all except three branches appear to be within 32 cells of an average branch length, 5 bits should be adequate for encoding most changes in successive node coordinate values. Upon allowing an average of 6 bits for this purpose, the 20 node coordinate changes plus 14 branch orientation bits require a total of 134 bits.

$n_4$bits---specify the number and locations in the node sequence of the start node of each branch to be displayed as a broken line. Again, assuming that the number of such branches is small, a very generous overbound on the number of bits required is $(n_1/2)$ log n_1. One bit is required here for the example.

Components Group

$n_5$bits---specify the element sequence corresponding to the specified node sequence. Assuming roughly 4 bits per branch element, the number of bits required here is about $4n_1$. Since about half the branches are just wire connections, the number of required bits will be considerably smaller. Thirty two bits are required for the example.

Labels Group

Point Sequence 2 Group

$n_6$bits---specify the number of symbol sequences occurring in labelling. A generous upper bound for the number of such sequences is the number of nodes in the circuit node sequence. Hence, 4 bits are adequate for our example. There are 9 symbol sequences there.

$n_7$bits---roughly 5 bits should be adequate for encoding the change in each coordinate in going from the first symbol in one sequence to the first symbol in the following sequence. Since there are 9 symbol sequences, about 90 bits are required here.

$n_8$bits---specify the common font size. 5 bits are adequate here.

$n_9$bits---with a maximum of 5 symbols in any symbol sequence, 18 bits are adequate to specify the numbers of symbols in each symbol sequence.

n_{10}bits---identify all the symbols in the nine symbol sequences. There are 20 symbols. If the symbol set includes all English upper and lower case letters, arabic numbers from 0 through 9, and mathematical symbols such as "=", "x", etc., then 7 bits is more than adequate to code each symbol. An average of 5 bits per symbol is probably attainable using a Huffman code since only about one third of the letters in the alphabet are apt to appear with significant frequency in circuit labeling. Hence, 140 bits are certainly more than needed here.

Arrows Group

n_{11}bits---specify the number of arrows. There are 2 in the example. Two or three bits is probably sufficient for the number of arrows to be enco tered.

n_{12}bits---are used for locating points for the arrow shafts. Three points would be used for one arrow and two for the other in the example. For each arrow, changes in point coordinate values are encoded, so about 6 bits per coordinate change should be satisfactory. Thus, about 60 bits are required for the example.

n_{13}bits---are used for specifying the numbers of heads for each of the first and second arrows and then to which end of an arrow the head is connected if an arrow has only one head.

In summary, about 200 bits are required for encoding the circuit elements and their connections, and somewhat less than 325 bits are required for encoding labels and arrows. Surprisingly, labels and arrows account for most of the representation code. If one were willing to build more intelligence into the system, the numbers of bits required for representing labels and arrows could be considerably reduced. For example, were the system able to recognize that labels B, E, and C are labels for the transistor base, emitter and collector, the system could encode the fact that the transistor terminals are to be labeled and not encode the individual symbols and their locations. If it were recognized

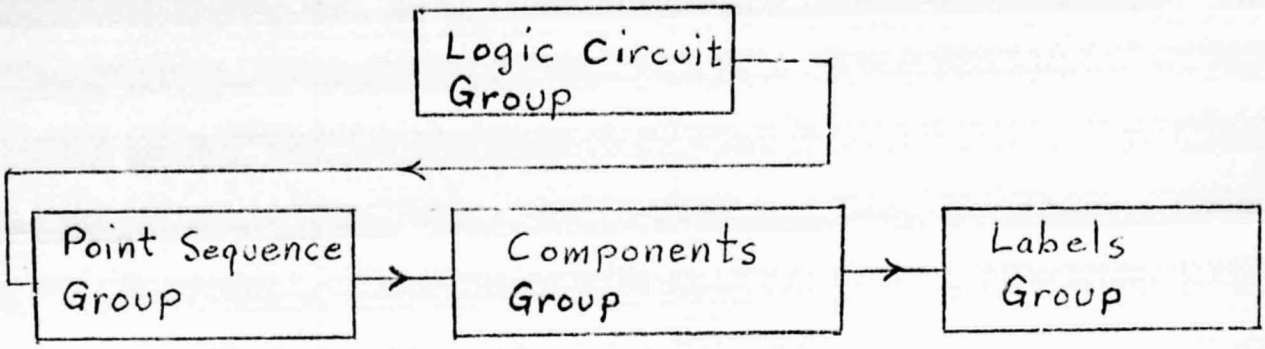
that the label "47k" is associated with the resistor between nodes 3 and 4, the first symbol could be located with respect to the local coordinate system for the associated resistor, and the location be sufficiently crude that some compression thereby achieved.

Finally, additional compression of the circuit component symbols and their interconnections is possible by not encoding node spatial location information. However, then the circuit branches with which labels and arrows are associated must definitely be understood. It might even be possible to obtain further compression by recognizing the circuit to be a grounded emitter amplifier, but the extent of this added compression is not clear and it would probably be difficult to extend circuit recognition to circuits containing two or more transistors.

2.2 Logic Circuits Group

(Combinational and Sequential)

The encoding here is similar to that for the Electronic Circuits Group except that arrows are usually absent. The decomposition of this group into subgroups is as shown.



First a node sequence (changes in node coordinates from node to node) and then branch elements are specified. Labels play an important role in these circuits. The components permitted are And, Or, Nand, Nor Gates, Inverters, Negating, Nodes, and J-K and S-R Flip-Flops. Also, more general modules represented as rectangular boxes with labels inside and perhaps outside. Since gates have two or more inputs and one output, we adopt the convention that gate input and output node sequences are encoded in the restricted format of input nodes sequentially ordered and output node either preceding or following the input nodes. A 1 or 0, respectively, following the component designation will indicate whether the input nodes precede or follow the output node.

Unlike other components, labels are an inherent part of the symbol for a flip-flop. The J (or S) and Q nodes must be so labeled and, when present, the CP (clock pulse) node must be labeled. Hence, we provide this information in the code for a flip-flop within the Components Group. In the Point Sequence Group for encoding logic circuit nodes, the nodes on one side of the rectangular flip-flop box will be given and then sequentially nodes on successive sides of the box. In the Components Group, after the flip-flop nodes have been identified the J (or S) node and Q node (along with the alternative symbol such as A, B, etc., which may be used in place of the symbol Q) are identified.

Modules such as adders, etc., are usually represented as a rectangular box with interior and exterior labels. The proposed code format here involves

encoding input and output nodes and labels and an algorithm is provided for constructing an appropriate rectangular box based on node data. Since nodes may exist on three and perhaps even four sides of the module rectangular box, nodes on one side of the box would appear as a subsequence in the total node sequence, and nodes on the remaining sides of the box would be encoded sequentially as they occur when proceding around the box in one direction. When identifying the node subsequence constituting a module in the Components Group, the nodes appearing on each side of the rectangular box must also be delineated. This can conveniently be accomplished through use of a sequence of three code words, each specifying the number of nodes on a side of the box.

Labels pose a greater problem here than with electronic circuits because the labels here are associated with nodes and these nodes are closer together than are the entities labeled in electronic circuits. Furthermore, the labels in logic circuits usually denote logical variables, and this information greatly facilitates the understanding of circuit behavior. Finally, certain labels should be contained entirely within a rectangular box. The burden of meeting these requirements will be on the pattern recognizer and parameter adjustment programs at the encoder. Once parameters have been adjusted to meet the constraints, the labels can be encoded as a Symbol Sequence Group as was the case with the electronic circuits.

An arrow head denoting direction of data flow may appear on a data line. This symbol could be treated as a circuit component or a label, but it is assumed here that it is treated as the latter.

2.3 Transistor Curves of i_C Versus v_{CE} Group

A set of curves occurring frequently either alone or with additional drawing such as a load line, is the set of curves of collector current versus collector-to-emitter voltage for various values of base current (see Figure 2). These curves can be encoded as a general Graph Group, but are highly stylized and therefore can usually be encoded somewhat more efficiently as an i_C vs. v_{CE} Graph Group. Parameter specification then is the location of the origin and the last points for the coordinate axes i_C and v_{CE}. Then the set of curves is specified. This can be done by representing a prototype curve, the one for smallest i_β is this case, by a suitable number of pcints (and an interpolation algorithm stored at the transmitter and at the receiver--perhaps a cubic spline) and multiplying the curve by a constant plus the integers 2 through n to obtain the n curves in the graph. Then axis labels and curve labels are specified. A reasonably complete set of situations would range from specification of all labels--i_β, i_C, v_{CE}, units and sets of values for these parameters--to use of a symbol subgroup where an arbitrary set of symbols can be provided.

Hence, the numbers of binary symbols involved is the following:

$n_1$bits---locate origin and end points of the i_C and v_{CE} coordinate axes. There are 4 parameters here.

$n_2$bits---specify curves. A 1 or 0 will specify whether curves are to be represented as multiples of a basic curve or are to be specified individually. The number of points to be used per curve is then given and finally a sequence of changes in x, y pairs per point. Four points to characterize the left side curved portion of a prototype and a right end point would probably be satisfactory. A straight line might be used for interpolation over most of the curve. Finally, for the multiples of a prototype case, the number of multiples and the multiplicative factors must be specified. Successive multiplicative factors which differ by a fixed increment--which is most likely to be the number 1--should prove satisfactory. Hence, 3 parameters to describe the multiplicative factors are required--the number of curves, the increment between successive multiplicative factors, and the first multiplicative factor.

$n_3$bits---encode labels. There are four reasonable situations here which can be named with 2 bits. These are: complete labeling consisting of i_β, i_C, v_{CE}, units (i.e., "m.a", "volts"), and values: i_β, i_C, v_{CE} only; complete labeling plus a Symbol Group; or finally, a Symbol Group alone. The Symbol Group permits variations but also additional labeling if loadlines or other additional curves are used. For the first case, values of i_β, e.g., i_β = .01 ma, i_β = .02 ma, etc., would be specified at each curve and values of v_{CE} and i_C would be reproduced at points along their respective axes. For each axis, the location of the first such value would be given, then the value given, and then changes in location and values of the remaining values to appear would be encoded. The total number of bits required for representation of this group is small.

3.0 Natural Less-Stylized Pictures

Because of inherent variability, there are few stylized group structures such as the Electronic Circuit Group here. Rather, the patterns of interest are unions of curve segments replications of line drawings or semi solidly black regions, and regions which have no line structure and are essentially all white or all black. Figures 3,4,5 are copies of much of the blackboard material appearing during two lectures in an undergraduate Histology course. Figures 3, 4 are drawn copies and Figure 5 a photocopy. Beyond compression which can be achieved by approximating line drawings with splines, we can achieve significant compression if: (1) a pattern contains considerable replication of a subpattern, (2) a pattern is a perturbation of an idealized simply parameterized pattern (which is another way of saying that a pattern is significantly more efficiently parameterized within a certain local coordinate system than within the original and, also, that the local coordinate system is efficiently described within the original coordinate system).

For the material in Figures 3, 4, 5 and for a wide range of other material, among the useful structure for efficient representation are the following. Curve segments are constructed from other curve segments and from point sequences. Point sequences are taken directly from data given with respect to a coordinate system, or are described with respect to curve segments or regions. Regions are built from curve segments. Replications are constructed in terms of a basic prototype pattern and a point sequence at which the basic pattern is to be replicated. As is clearly exhibited in the drawings in Figures 3, 4, 5 randomness plays an important role in trying to model replication. Each of the structures--Curve Segment Sequence, Point Sequence, Region and Replication--will be modelled as a group. Since there may be two or more of one or more of these group types, in addition to a group type identifier we will also need to name the various groups within each type. Before going into further detail on the structure of these groups, we briefly indicate how they would be used for the representation of a few of the subpictures in Figures 3, 4, 5. Consider first Figure 4a. The node numbers there are not part of the picture to be reproduced but rather are for facilitating description of the encoding procedure. First, a Curve Sequence Group representing the three nested ellipsoidal type curves would be established. (We will refer to this group as Group 1). The three curves are not perfect ellipses, but rather small perturbations of perfect ellipses. We therefore have a choice. Either the curves can be represented accurately--or a random

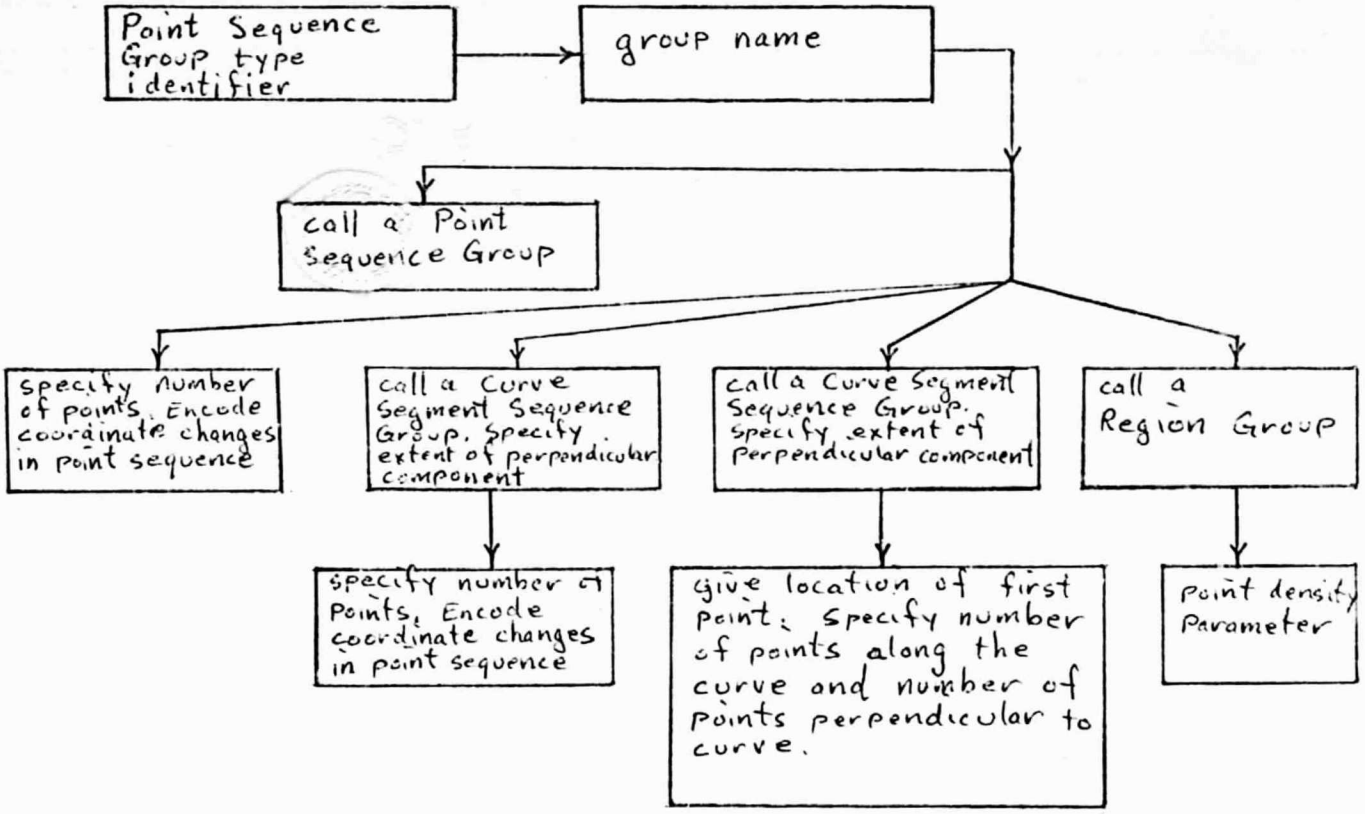
mechanism can be employed for slightly distorting the best fitting ellipses. (The receiver and the transmitter would have the same random number generator.) The latter result would probably be worse reproductions than the best fitting ellipses, but the approximations would retain the naturalness and nature of the original data. This is an important point. We can produce an exact reproduction if this is necessary. But for concept teaching, a rough reproduction but of the same character is usually as satisfactory and can be encoded with fewer bits and require less processing. Fifteen spatial parameters are needed in encoding the three ellipses. (The use of circles instead of ellipses here might require fewer bits to achieve the same accuracy in curve fit.) Next, the end points of the straight lines are encoded as a Point Sequence Group. The points would be encoded by using the approximation appearing in Group 1. The points 1 through 8 would be specified with respect to the outer perturbed ellipse approximation appearing in Group 1, and then the points 9 through 15 would be specified in a similar way with respect to the adjacent perturbed ellipse approximation. The Point Sequence Group would thus contain a description of the locations of a sequence of 15 points (we will refer to this group as Group 2). More specifically, the points 1 through 8 could be located exactly by treating the ellipse as a local curvilinear coordinate system with a point located in accordance with its component along the curve and then its component perpendicular to the curve. Since the latter component will be essentially 0 here, we ignore it and just encode along curve distance of a point from the preceding point. We again have a choice since the points appear to be roughly equi-spaced. We can encode exactly the perturbations from an average value in their spacing or we can define a simple randomization mechanism which perturbs them slightly from their average spacing. The reproduction in the latter case would differ slightly from the original, but the picture would be of the same character. Next, we construct a second Curve Sequence Group. (Refer to it as Group 3 .) This group contains all the straight line segments such as that connecting nodes 1 and 9 and that connecting nodes 2 and 9, etc. Group 3 uses Group 2. Nodes in Group 2 are numbered as in the figure and the straight line segments are characterized by a sequence of pairs of node numbers. Finally, the black region must be modelled. A Region Group (refer to it as Group 4) is constructed by specifying the sequence of four curve segments which bound the region. The two straight line segments can be retrieved from Group 3. The two perturbed elliptic arcs can be obtained from Group 1. Since the arcs do not appear

explicitly there but rather are segments of longer curves appearing there, the arcs must be derived through operations on the curves in Group 1. In Group 4 the required segments are extracted through retrieving the two perturbed ellipses from Group 1, retrieving the desired curve segment end points, points 1, 8, 9, and 15 from Group 2, and obtaining the curve segments from this data. Finally, the black region can be obtained by calling on a group realizing a set operation--complementation on the region specified in Group 4 in our example. Consider Figure 4b next. A reasonable encoding here is as follows. First, specify a point sequence for curve 1-3. It is not clear whether the most efficient code would result from approximating the data by a circle--requiring 3 parameters--and then specifying point locations with respect to this universe, or whether it would be better to locate points with respect to the coordinate system for the entire field. Call the resulting group, Group 1. Next, Group 1 is used in a Curve Segment Group to approximate curve 1-3. We refer to this group as Group 2. Then curve 1'-3' is encoded as being at constant perpendicular distance from the curve of Group 2 over the intervals 1'-2' and 2'-3'. We refer to the Curve Segment Sequence in which curve 1'-3' is encoded as Group 3. Then a point sequence is encoded for curve 4-5 as Group 4 and a Curve Segment Group then established, Group 5. Using the Replication group, the adjacent three curves are encoded as shifts of the basic pattern, curve 4-5. Finally, the Purkinje cell sequence and Granular and Molecular layers must be encoded. For example, the Purkinje cells can be encoded using the Replication Group as follows. First, we determine a curve through the cell locations and, therefore, with respect to which the cell locations can be efficiently given. Toward this end, specify a sequence of about 5 points with respect to curve 1-3 as a coordinate system. These points are to be equispaced along curve 1-3 and have the appropriate perpendicular distances from curve 1-3. Few bits are necessary here. Call this Point Sequence Group, Group 6. Group 7, a Curve Segment Group, is defined in terms of Group 6 and is used as a coordinate system for locating the replication points for the Purkinje cell set. The basic Purkinje cell is relocated at equispaced intervals along the curve of Group 7 and is oriented perpendicular to the curve. Some simple random mechanism can be described for modifying the basic Purkinje cell as replicated at each location. Replication in the vicinity of curve 1-3 of the other cell types can be handled in a similar way.

The various groups are constructed as follows.

3.1 Point Sequence Group

This group assembles a sequence of points for use in another Point Sequence Group, or in a Curve Segment Sequence Group, or in a Replication Group, or perhaps others not as yet defined. Points are assembled here by calling one or more other Point Sequence Groups and/or by calling one or more local coordinate systems and specifying points with respect to these systems. If possible, the points are encoded in the order they will be needed by one or more of the other groups constructed for a picture.



More specifically , a subsequence in the Point Sequence Group is encoded in one of two ways, these being (1) by calling an already formed Point Sequence Group, (2) by calling an appropriate coordinate system and encoding the points with respect to this system. Four coordinate system structures appear to be worth distinguishing. The first is simply the rectangular coordinate system for the entire picture. The second and third are curvilinear coordinate systems. The appropriate curve is specified by calling a Curve Segment Sequence Group, and the extent of the universe in the direction perpendicular to the curve is given. Points are encoded within the first-two universes by encoding the change

in coordinate values of two successive points. The third system is to be used for points which are equispaced in an array, or more generally, are equispaced along the coordinate curve and equispaced perpendicular to the coordinate cruve. Then the location of the first point in the sequence is given, as are the numbers of points parallel to and perpendicular to the coordinate curve. The many small circles in figure 5 are an example of this. This encoding can result in many fewer bits than would be required by using one of the first two systems. The fourth system is a region accessed by calling a Region Group. The region is specified within the Region Group as a curve segment sequence constituting a closedboundary curve. This system would be used for rather arbitrary region boundaries and points which appear to be randomly distributed throughout the region. Point locations would not be duplicated within the region, but rather the random process identified and a generator for a similar one used to generate the points within the region. For the most part, we assume a two-dimensional poisson process would be appropriate and therefore merely encode the point-density parameter. Receiver and transmitter would have identical poisson process generators.

A one bit code, following the Point Sequence Group header and then group name, could specify whether a subsequence is to be encoded by simply calling an already formed Point Sequence Group or by encoding the subsequence with respect to one of the four described coordinate systems. Then two bits would specify which one of the four coordinate systems is to be used. If the choice is one of the last three, a few bits are required for naming the appropriate Curve Sequence Group or Region Group, and in the case of the Curve Sequence Group, a few bits are required for describing the extent of the coordinate system in the direction perpendicular to the curve. Code for the encoding of points within the first two coordinate systems is straightforward. First, the number of points within the sequence is specified. Then, changes in x and y cordinates in going from one point to the next are specified.

Sequences of subsequences as above are encoded in order to construct a Point Sequence Group. The code as described above is such that the decoder can determine when the code for one subsequence finishes and the next begins. Some further saving in the number of bits required is possible if, for example, two or more successive subsequences are encoded in the same type of coordinate system or are all called as already formed groups, but the saving is small and we therefore do not pursue the matter further here.

3.2 Curve Segment Sequence Group

We provide for a curve segment generation in one of four ways in this group. First, an already formed Curve Segment Sequence Group can be called in its entirety, or such a group can be called and one or more curve segments retrieved from it. Second, a primitive, e.g., a circle or a segment of a circle can be called. Third, a Point Sequence Group can be called or a subsequence called. Finally, a Point Sequence Group can be called and all or a subset of its elements rearranged in the order in which they are to be used in the curve segment determination.

More specifically, consider the first option. Then 2 bits will specify whether the entire Curve Segment Sequence Group is used, whether a subsequence of the curve segments is used, or whether a subsegment of a curve segment is used. If the second choice is made, the coding involves specifying the positions within the curve segment sequence of the first and last segments to be retrieved. If the third choice is made, the position within the curve segment sequence of the curve segment to be extracted is specified, and then the first and last endpoints delimiting the curve subsegment of interest are given. The coordinate system within which the two points are encoded is the same system within which the points determining the curve segment are encoded.

If the third option of calling a Point Sequence Group is chosen, one bit specifies whether the whole group or a subsequence is used. If the latter, the positions of the first and last points in the subsequence are specified.

If the fourth option of calling a Point Sequence Group and rearranging all or a subset of the points is chosen, one bit specifies whether all or a subset of the points in the group are used. If the latter, then the number of points to be used is specified and a sequence of position labels for the point subsequence given. These labels are given in the order in which the points are to be considered in constructing a curve segment with splines. For example, if the original point sequence were $x_1, y_1, x_2, y_2, \ldots, x_n, y_n$, then the subset rearrangement $x_2, y_2, x_1, y_1, x_5, y_5$ would appear in the new Point Sequence Group as 2, 1, 5.

An appropriate encoding of the second option is obvious.

3.3 Replication Group

This group serves to reproduce a basic pattern at a number of locations. The basic pattern is represented in a rectangular local coordinate system with axes parallel to those of the original coordinate system for the entire picture. This local coordinate system and basic pattern must be encoded. The basic pattern may be a line drawing which can be represented as a Curve Segment Sequence Group, or it may be a pattern with considerable black area in which case the original data comprising the pattern may be used as is without modification or it might be compressed somewhat using cluster coding. (Note that even though we are dealing largely with line drawings, it is easy to draw small subpictures with considerable black area and provision must therefore be made to handle such patterns.) The Purkinje cells in Figure 4b are an example of small patterns with considerable black area. In general, we expect the basic pattern will be reproduced at each location with a fixed orientation in the original or in the coordinate system, with respect to which the relocation points are described. For example, the first axis of the rectangular coordinate system within which the basic pattern is encoded might be maintained at a constant angle with respect to the first axis of the curvilinear coordinate system for specifying the relocation points. Again, this is a good model for the Purkinje cell replication in Figure 4b. Finally, for those situations where the basic pattern is not replicated with constant orientation the sequence of changes in orientation must be given. Here one bit can indicate whether the coordinate system within which orientation is encoded is the original system or the one used for specifying relocation point coordinates. Then, the sequence of changes in orientation encoded.

In greater detail, the code is determined as follows. Following the Replication Group type specification and encoding of the group identifier, the basic pattern is encoded. One bit specifies whether the pattern is called as a Curve Segment Sequence Group or is a reproduction of a subset of the original picture data. If the latter, then some appropriate local rectangular coordinate system with axes parallel to the original system is specified for the pattern. There are 4 parameters describing this coordinate system--the extent of the universe in the x and y directions and the location of the local origin within this universe.

Next, the relocation points are to be encoded. This is handled by calling a Point Sequence Group. Finally, basic pattern orientation must be specified

at each relocation point. Two bits convey which of three possibilities pertain. (1) Fixed orientation with respect to the original coordinate system for the whole picture. (2) Fixed orientation with respect to the coordinate system used in the Point Sequence Group. (3) Varying orientation with respect to the coordinate system used in the Point Sequence Group. Encoding for (1) and (2) merely requires encoding the fixed orientation. Encoding for (3) involves encoding the sequence of changes in orientations of the basic pattern from one relocation point to the next.

3.4 Region Group

The structure of this group is trivial. Following group type specification and then group identifier, a Curve Segment Sequence Group is called. The curve segment sequence should be a simple closed curve which does not intersect itself. Since the curve is a directed curve, one additional bit specifies whether the region interior is on the right or the left when traversing the curve in the forward direction.

One or more additional groups can be defined which take as arguments Point Sequence Groups, Curve Segment Sequence Groups and Regional Groups and realize set operations such as complementation, intersection, etc.

3.5 Examples

We estimate the numbers of bits necessary to encode figures 4a and 4b. There is no obviously best coding although if using the groups given and essentially in their described form, a coding close to the best can be readily determined. A few new options and modifications in defined groups are seen to be clearly desirable as one tries to encode more pictures. The encodings described in the remainder of this section differ somewhat from those, for the same pictures, briefly summarized in section 3.0. The encodings here use one or two fewer options than would be necessary for the others and are therefore simpler to describe. The required bit estimates here can be considered to be upper bounds on what can be achieved using the best of the encodings here and in section 3.0.

Figure 4A

We encode Figure 4a somewhat differently here than as described in section 3.0. The figure encoded here is the three closed curves and the associated lines and black area, and also the arrows and underlined letters CSF. The compression realized is about 20 to 1. As discussed at the end of this section, if a less exact reproduction is permissible, greater compression is possible

Since only a few group types and at most a few instances of each group type appear in the encoding of these examples, 10 bits is more than adequate for specifying a group type and naming an instance. First a Point Sequence Group is established and the coordinate changes encoded for the sequence which we describe in terms of our point labels: 1, 9, 2, 3, 10, 11, 4, 5, 12, 13, 6, 7, 14, 15, 8, 1, 9 and an additional 7 points for the inner circle. We assume that these points are adequate for representing the three closed curves. If we assume that 10 resolution cells are required for representing the small closed curve, then a square 128 by 128 0-1 character matrix would be needed for encoding the whole picture. This field consists of 16,384 bits. Hence, less than 10 bits are required for specifying group type and naming the group, 4 x 7 = 28 bits are required for determining the local rectangular coordinate system parameters, and then about 23 x 10 = 230 bits are necessary for encoding the point sequence coordinate changes. There are 23 points here and we assumed about 5 bits per coordinate change. Thus, about 270 bits are required for this group.

Next, a Curve Segment Sequence Group is established for the three closed curves. Assume an option not contained in the definition of this group in section 3.2, namely, when using a Point Sequence Group a subsequence can be extracted by associating a 1 or a 0 with each point depending on whether or not the point is to be in the subsequence wanted. Then less than 10 bits are required for specifying group type and name, less than 10 bits are required for calling the Point Sequence Group for each of the three closed curves, and 23 bits are required for extracting the point subsequence to be used for each closed curve. Thus, roughly 35 bits per curve are required here, or about 120 bits for the group.

A group which we haven't defined would be used for encoding the 8 straight lines, namely, a Line Sequence Group. This group simply directs the receiver to connect straight lines between successive points in a Point Sequence Group. The option would be available of having one binary symbol associated with each point in the Point Sequence Group called with a 1 indicating that a point be used and a 0 indicating that it not be used. Hence, less than 10 bits specify the group type and name, 10 bits are required for calling the Point Sequence Group and 23 bits are required for extracting the point subsequence to be used. Roughly 35 bits are required here.

Finally, for the black area in the figure a Region Group type and name are specified, using about 10 bits. And 10 bits are required for calling the Curve Segment Sequence Group defining the region. This Curve Segment Sequence Group is as follows: 10 bits specify the group type and name. To incorporate the line segments between points 1 and 9 and points 15 and 8, 10 bits are used to call the Line Sequence Group and 8 bits are used to extract the two line segments of interest. Hence, about 20 bits are required to specify the two line segments. The two curve segments between points 15 and 9 and points 8 and 1 are incorporated as follows. Curve Segment Sequence Group 1 is called, using 10 bits, and for 4 bits the first option is chosen -- calling a single curve segment. Then points 9 and 15 are specified using 14 bits for the first point and 10 for the second. This is repeated for the other curve segment. Hence, about 70 bits are used in calling the two curve segments. Consequently, this Curve Segment Sequence Group is encoded for 100 bits. Finally, assume the Region Group contains an option for performing set operations such as complementation of the 0's and 1's within the region. Then the total number of bits required for encoding the black area is roughly 125 bits.

Finally, the three arrows, the letters CSF, and the two underlines must be encoded. Roughly 95 bits would be required to use the Arrows Group defined in Section 2.1. Roughly 60 bits would suffice for encoding CSF using the Symbol Sequence Group, and roughly 80 bits would be required using a Point Sequence Group and then a Line Sequence Group to encode the two lines under CSF.

In Summary, Figure 4a can be encoded in less than 800 bits in the way described. This provides a compression of 20 to 1. Substantially greater compression is possible if the following approximation is satisfactory: (i) The three closed curves are approximated as small random perturbations of ellipses. (ii) The points 1 through 8 and points 9 through 15 are represented as small random perturbations of uniformly spaced points around the closed curves on which they lie.

Figure 1b

A 256 x 256 cell field is required for this picture. We encode the fibers first. 10 bits specify a Replication Group and its name. Then a local coordinate system for the curve between points 4 and 5 is specified and encoded using 4 x 16 = 64 bits. Five points are more than adequate for specifying the curve and require about 50 bits at about 10 bits per point. Finally, the 4 points at which this curve is to be replicated can be specified using less than 48 bits where we have assumed about 12 bits per point. Thus, the 4 fibers can be encoded using fewer than 125 bits.

Next, the two long arcs between points 1 and 2 and points 1" and 2" are encoded. A point sequence of 10 points can be established for each curve. This requires about 130 bits per curve or a total of 260 bits. When used in a Curve Segment Sequence Group, the number of bits required to specify the two curves is about 300 bits.

We then encode the Purkinje cells. Assume an area of about 100 pixels (picture elements) is required to represent a prototype Purkinje cell. With 4 x 16 = 64 bits to represent the local coordinate system for the cell, a total of about 170 bits are required for encoding the prototype. The 10 points at which the basic cell pattern is to be replicated are encoded almost as described in section 3.0. The exception is that the 5 points in the Point Sequence Group denoted group 6 there are specified in the coordinate system for the entire picture rather than in a coordinate system determined by curve 1-3. This change simplified the encod- without requiring additional bits. Hence, about 65 bits encode group 6. About 25 bits encode group 7 which defines a curve determined by the points in group 6,

and the locations of the 10 equispaced points along this curve and at which the prototype Purkinje cell is to be replicated are encoded as a Point Sequence Group using roughly 30 bits. Thus, roughly 120 bits specify the locations and orientations of the 10 Purkinje cells. 300 bits should be sufficient for specifying the Replication Group for these 10 cells.

The molecular and granular layers are then encoded in an obvious way. The points at which the basic patterns are replicated appear to be randomly distributed over a region here. This region is conveniently defined with respect to the coordinate curve through the Purkinje cells, i.e. the curve specified in group 7. 800 bits should be more than adequate for encoding these two layers.

Finally the 4 words and associated 3 straight lines must be encoded. The 4 words can be encoded as a Symbol Sequences Group, using roughly 280 bits, and the 3 straight lines can be encoded in a Straight Line Sequence Group using less than 170 bits.

In summary Figure 4b can be accurately encoded using fewer than 2,000 bits. With 256 x 256 = 65,536 bits required for the uncompressed picture, this represents a compression of better than 30 to 1.

By exploiting acceptable variability in reproduction and using structural knowledge of the cells and tissue being dealt with, additional compression is possible.

4.0 Pattern Recognition

At first glance, the pattern recogniation required to realize the suggested encoding of the material appearing in the electronics course or in the histology course seems extremely formidable. However, upon closer examination it appears to be within the realm of the practical. Appropriate pattern recognition algorithms would have to be developed, but this problem seems to be manageable. We assume that reasonable care would be exercised by the communicator in drawing the pictures.

If x, y coordinate data is available as the picture is drawn (which would be the case when an electronic blackboard or a data tablet is used), the pattern recognition problem is of course very much simpler than when such sequential information is not available. We assume the latter, i.e., the worst case of the recognizer being presented with a complete picture. At present, there are effective handprinted character recognizers commercially available, and this technolcgy will develop rapidly. Hence, the cost of recognition of alphanumeric characters is or is about to become very reasonable. Distinguishing among the three subpicture group types introduced for the electronics course material or other subpicture types which might arise there seems to be a problem which can be managed. Also, once one of these subpicture groups has been type identified, low cost recognition of the subpicture parts is possible. For example, consider recognition of the parts of an electronic circuit schematic. There is great variability in hand drawn symbols for resistors, inductors, capacitors, and diodes, and these symbols are complicated. However, they differ from one another sufficiently that the data can be reduced through use of information--destroying transformations and this transformed data used for reliable recognition. Projection of the two dimensional binary cell patterns onto two orthogonal one-dimensional subspaces (i.e., projection onto orthogonal lines) results in reduced data from which the four symbols can be reliably and inexpensively recognized.

Figure 6 contains binary cell arrays of circuit symbols inputted through use of an acoustic data tablet. The print-out is somewhat distorted because of different scales in the horizontal and vertical directions. Figures 7a,b,c,d are projections of the 0, 1 arrays on the horizontal and vertical axes. Notice that when the elements are oriented horizontally (or vertically) they are readily distinguishable from their projections. Further study would be necessary to determine whether these projections contain sufficient information for

recognition when the element orientations is roughly 45° to the horizontal. However, the problem can always be circumvented by using two sets of projections for recognition--projections on a horizontal and vertical pair of lines and projections on a pair of lines making angles of 45° and 135° with the horizontal.

The required recognition for the tissue and cell pictures also seems reasonable. Long smooth curves must be recognized in Figures 3,4. This is readily accomplished through curve following where in passing through an intersection the path having the smallest change in curvature is the one followed. Replications of a basic subpattern such as the Purkinje cells or others in Figure 4b, or the circles in Figure 5 can be tentatively recognized by recognizing the existence of a high density of small subpatterns.

5.0 Further Work

The results on this work to date show that very high compression of line drawing data is theoretically achievable and that the pattern recognition required to implement this at or at close to these compression levels appears to be possible. What remains to be done is to refine the definitions of a few of the groups, define a few additional ones and experiment with a partial implementation of a coder and decoder in software for basically one subject area. This experimentation would permit a better understanding of the whole subject and also aid in the development of some understanding of what might be practical to implement in an operational system.

Subject material which would be suitable for a partially implemented system in software might consist of much of an introductory textbook such as "An Introduction to Electronics" by Oldham and Schwartz, the IEEE Transactions on Computers, and some electronic systems conference picture material. Three sets of problems require attention, these being pattern recognition, encoding in binary strings, and reconstructing pictures from their codes. Within each group, problems of interest in increasing order of difficulty are listed.

Pattern recognition. (i) Recognize that a picture constitutes an electronic or logic circuit schematic. Recognize the electronic and logic symbols and the wires in the diagrams. (ii) Recognize the portion of a circuit with which a label is to be associated. This may be based on proximity or on a low level of understanding of meaning. For example, the symbols Ω or v in the label indicate the label is to be associated with a nearby resistor or voltage source, respectively. The value of the number in a label provides a clue to its meaning; When dealing with a transistor circuit, a number in the thousands in the picture is highly unlikely to be a voltage and would most likely be a resistance. (iii) For pictures of natural phenomena, recognize that certain curves and areas are to be reproduced exactly, and recognize that certain other curves and areas need be reproduced in character only. (iv) For pictures not carefully drawn, use sequential stroke information in order to do the recognition rather than recognizing a completed picture based only on its 0-1 character matrix. This sequential information would be most easily obtainable if the drawings were made on an electronic blackboard or on a data tablet where the signal would be available as it is generated. (v) Recognize which subpictures should be specified with respect to other subpictures.

Coding. (i) Finalize the useful primitives and groups for electronic and logic circuit schematics and implement an encoder in software (ii) Examine the question of fewer more complex groups versus more but simpler groups and greater flexibility in encoding. The matter is posed in greater detail in 1 . later in this section. If one is interested in the problem of picture storage, then data base design considerations become important. (iii) Extend (i) to a broader class of subject material.

Picture reconstruction from codes. (i) Reproduce electronic and logic circuit schematics using fixed font symbols. (ii) Reproduce parametrized primitives such as conic curves, and reproduce piecewise linear approximations to more variable curves.

A number of the more general questions which need to be understood are the following:

(1) For representing the drawings of natural phenomena in the histology lectures, we used hierarchies of groups, where, for example, a group such as a Point Sequence Group might call another group which in turn might call other Point Sequence Group. Each new group introduced necessitates the transmission of the group type identifier and the group name and other information describing coordinate systems, etc. Furthermore, the group identifier and name is transmitted again when called by some other group. This approach permits flexibility but may very well require more bits than would be the case if more complicated groups were used. For example, the Electronic Circuit Group uses a restricted form of a Point Sequence Group and also encodes the point location information directly rather than calling a previously encoded group by type and name. The latter saves bits and computer processing but the groups may be more difficult for a human to design. We have some feeling for the tradeoffs here, but additional work is necessary.

(2) It would be easiest to implement a system for a subject using rather stylized drawings as is the case of the electronic lecture material. A better understanding of the general procedure and difficulty of designing groups for such classes would be desirable. What can be said about the numbers of required groups to be expected for other subject areas?

(3) For the looser structure involved in encoding the cell and tissue drawings, recognition of situations affording good compression is necessary.

That is, there are a number of different ways the picture can be encoded and the system should be able to recognize one of the better ones without having to try all and then choose the best. Is this feasible or is human intervention in this decision making required in many situations?

Acknowledgment

David Cooper is very appreciative of a number of most useful discussions with and suggestions by Dr. Thomas Lynch during the course of this grant.

References

[1] I. E. Sutherland, "Sketchpad, A Man-Machine Graphical Communication System", Proceedings of the Spring Joint Computer Conference, 1963, pp. 329-346.

[2] L. D. Davisson, "Universal Noiseless Coding", IEEE Transactions on Information Theory, Vol. IT-19, pp. 783-795, Nov. 1973.

[3] D. B. Cooper, "Feature Selection and Super Data Compression for Pictures Occurring in Remote Conference and Classroom Communications", record of the Second International Conference on Pattern Recognition, Lyngby-Copenhagen, August 13-15, 1974, pp. 111-115.

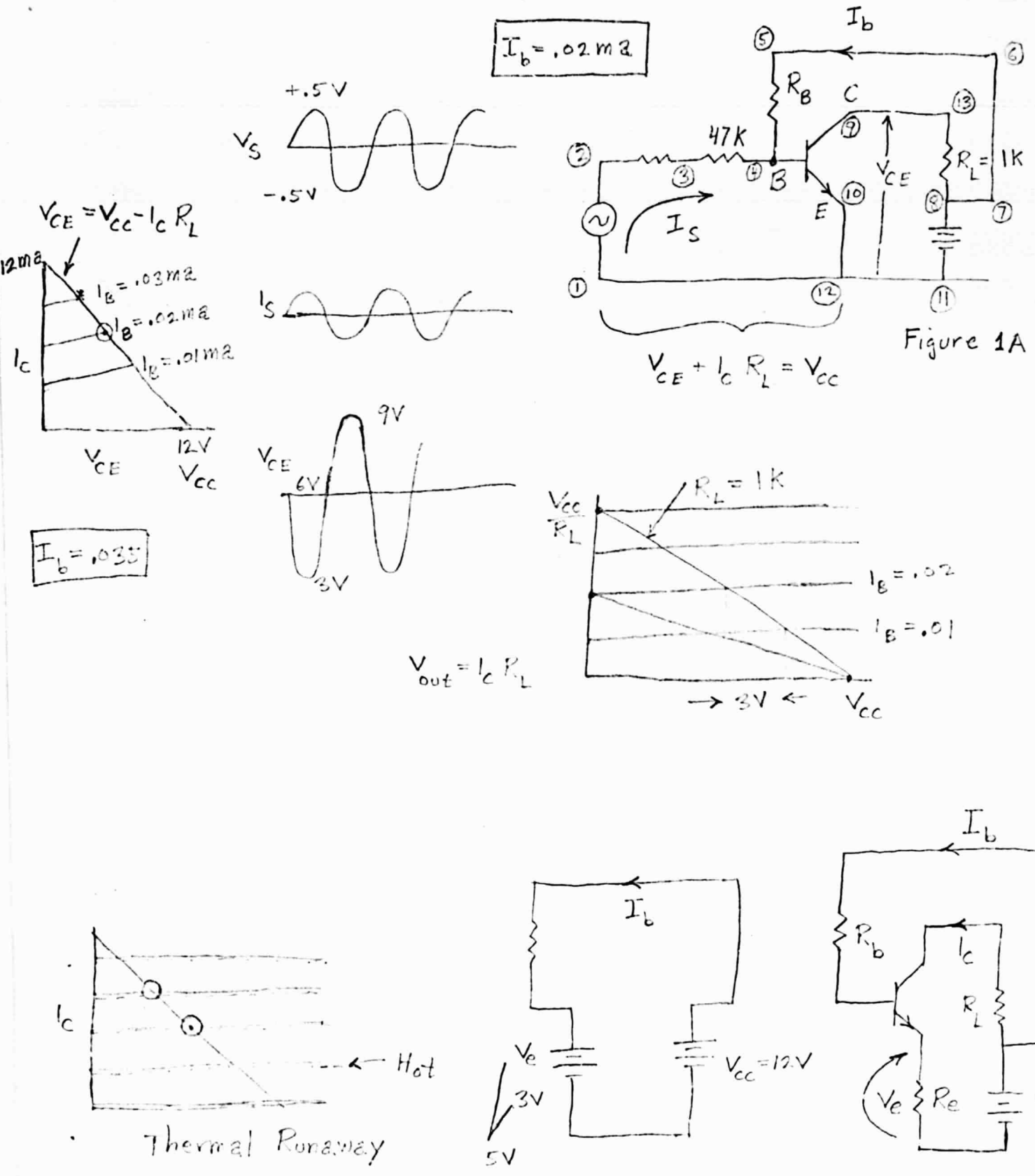


Figure 1A

Figure 1.

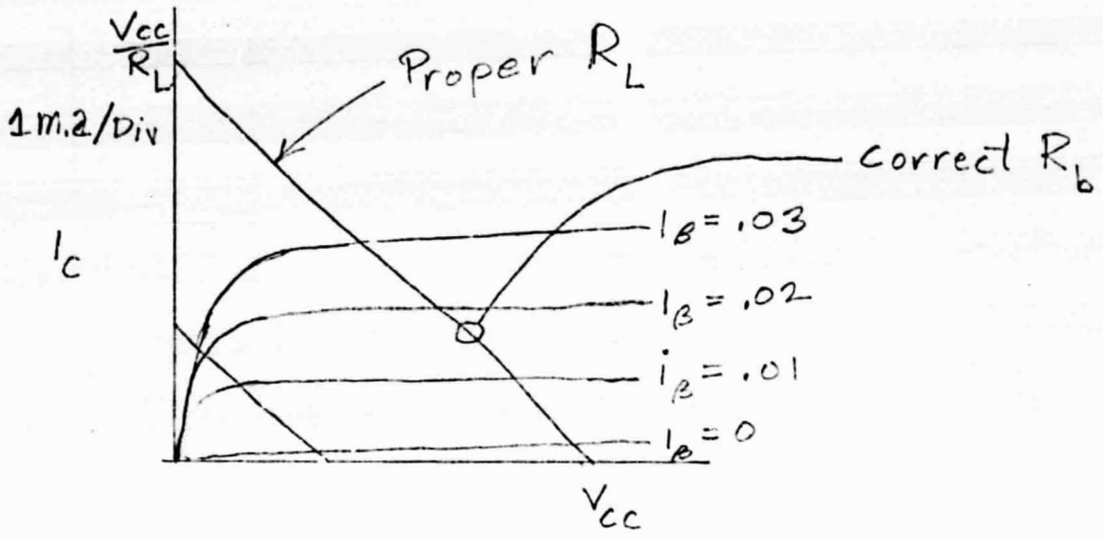
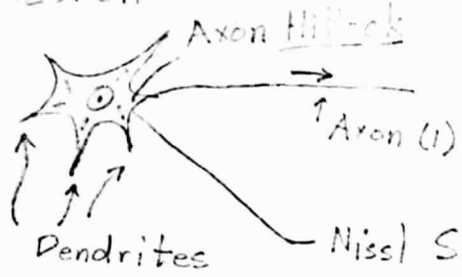


Figure 2.

Nerve — Nervous System

Physical	Fund	Irritability
CNS	Sensory	conductivity
PNS	Motor	
ANS		

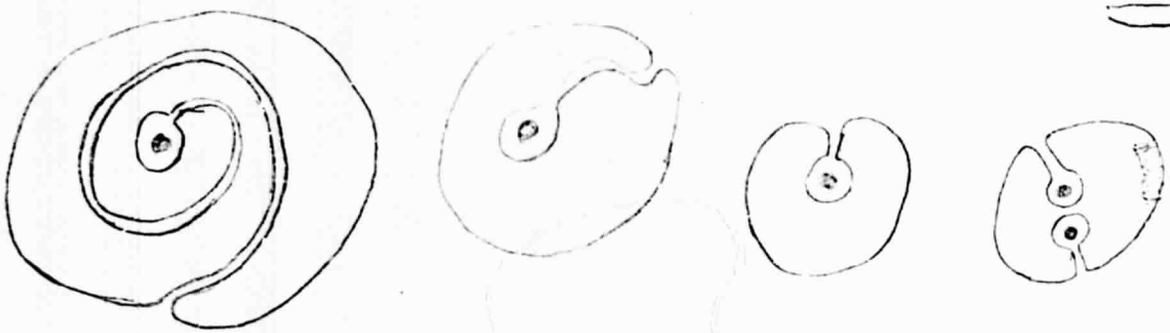
Neuron

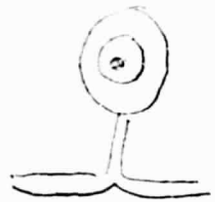


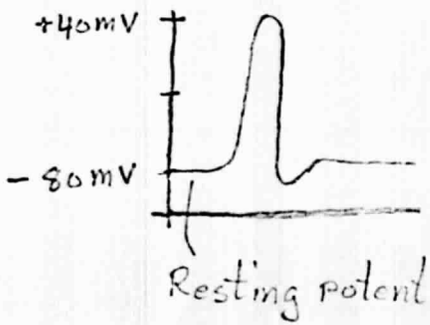
Bipolar

Neurolemma Sheath

Myelin







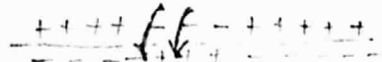


Figure 3.

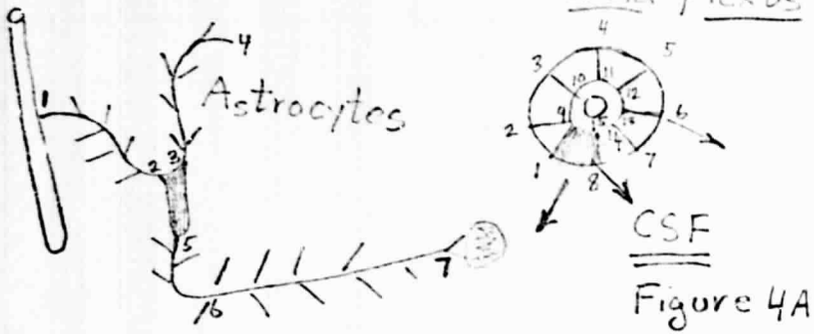
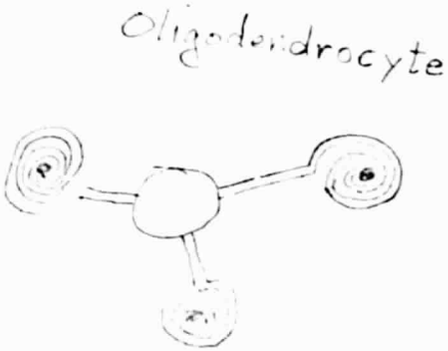


Figure 4A



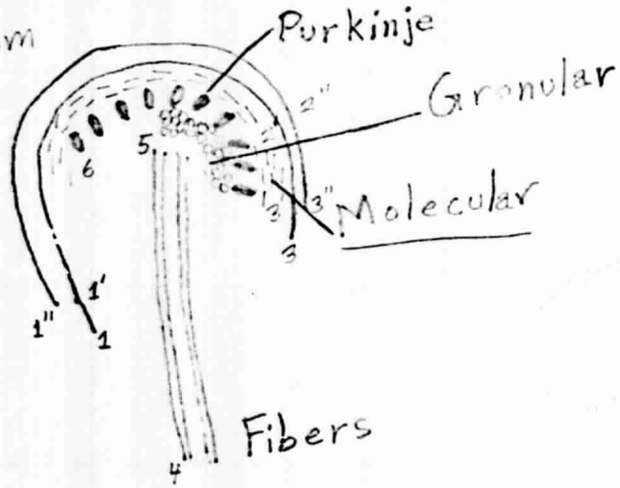
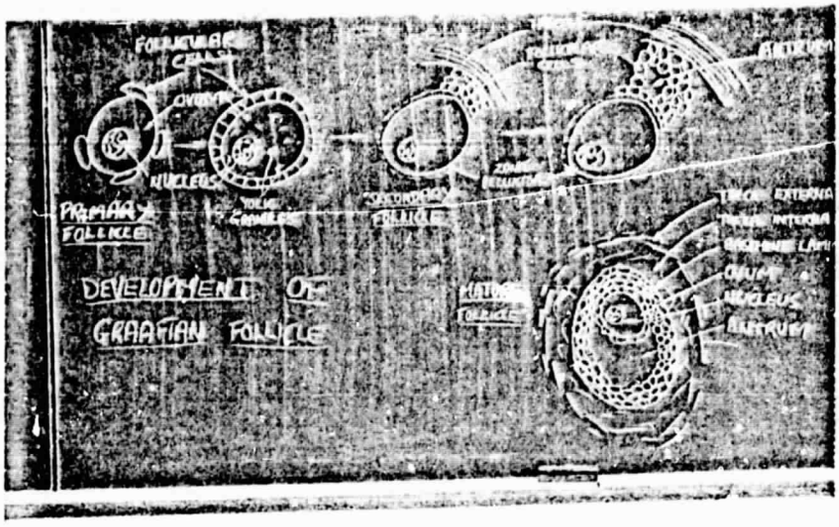


Figure 4B

Figure 4.



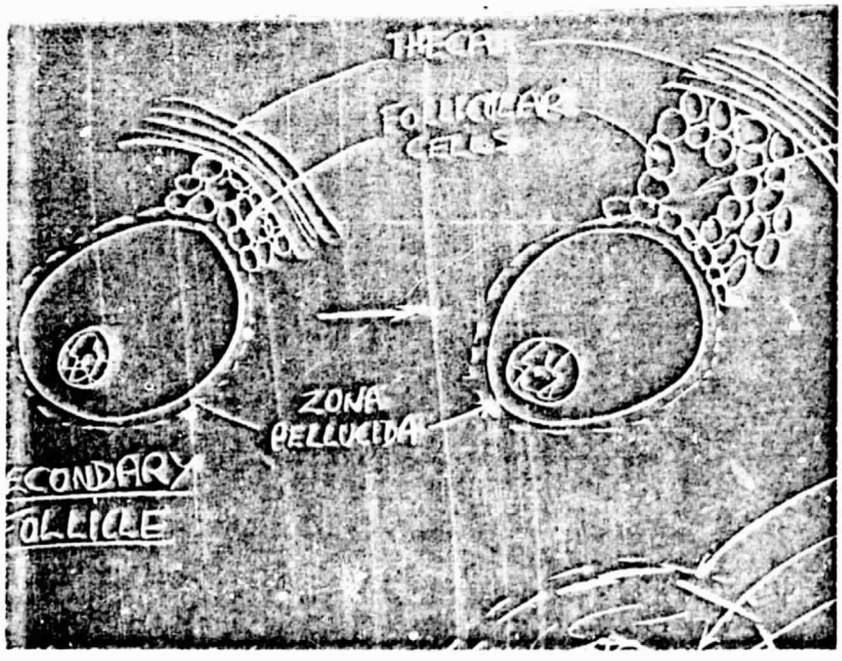


Figure 5.

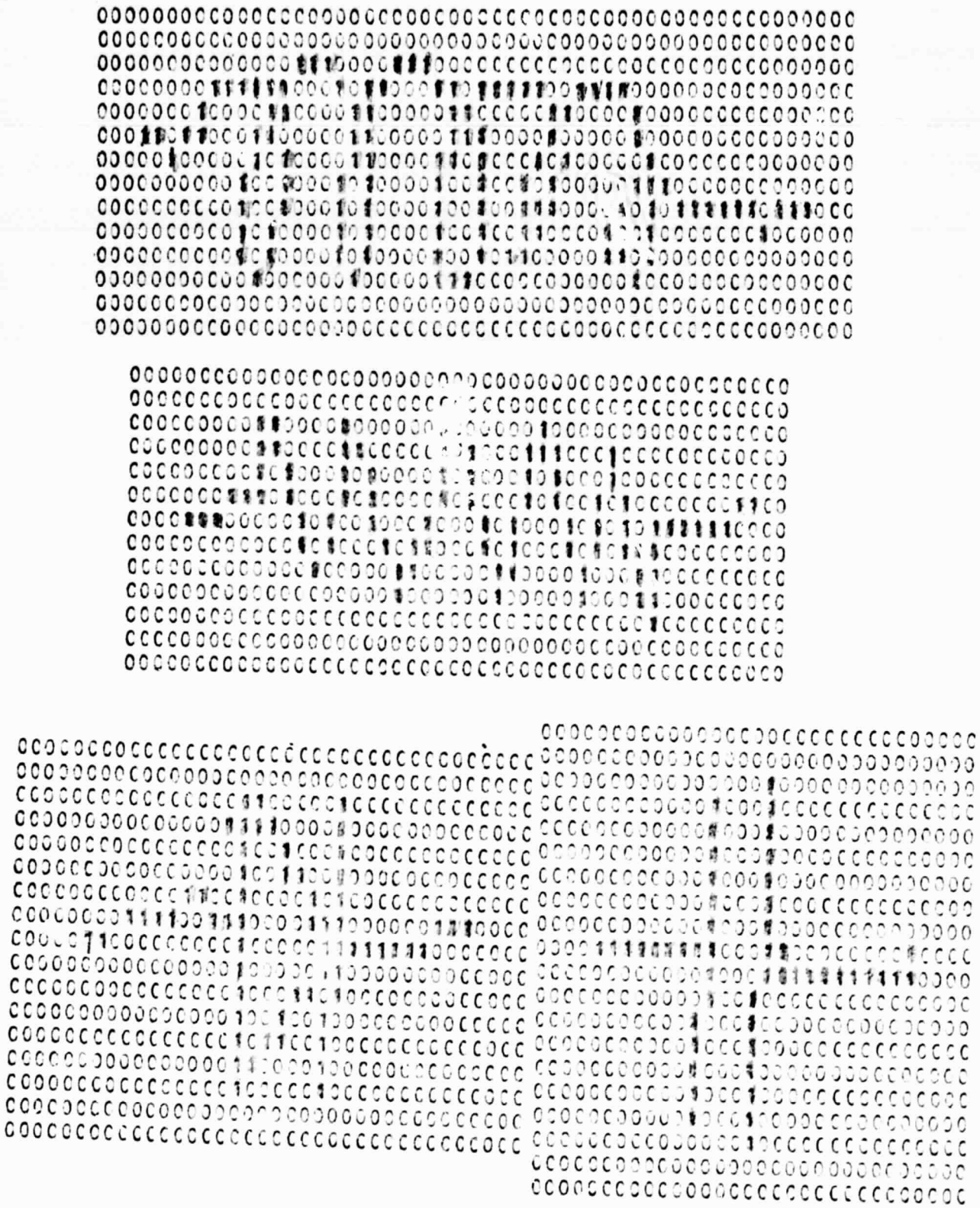


Figure 6. Printout Of Symbols Entered With The Acoustic Data Tablet

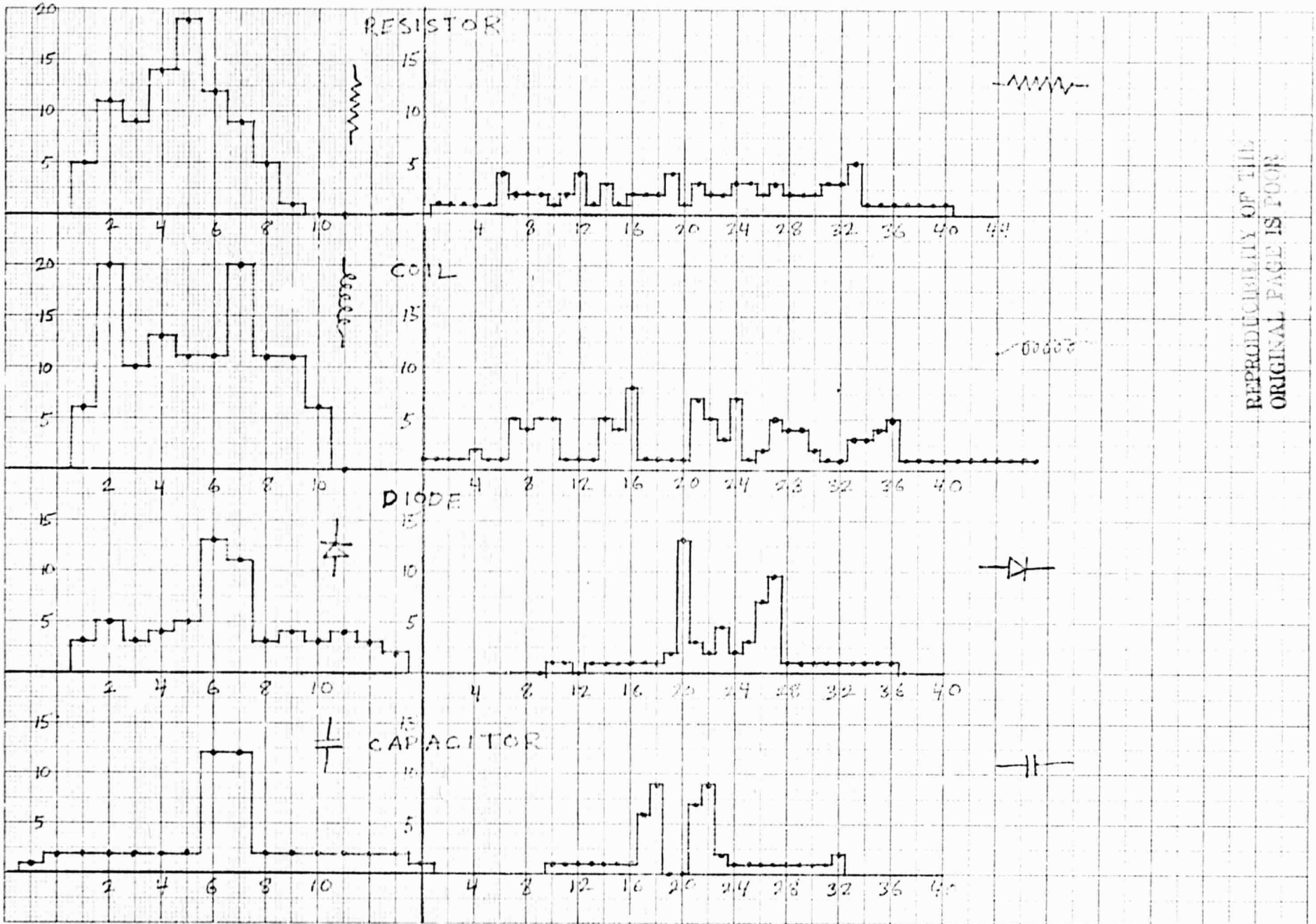


Figure 7. Projections On The Y And X Axes Of The 0-1 Arrays In Fig. 6

Proceedings of the Second International Conference on Pattern Recognition, Lyngby-Copenhagen, August, 1974

FEATURE SELECTION AND SUPER DATA COMPRESSION FOR PICTURES OCCURRING IN REMOTE CONFERENCE AND CLASSROOM COMMUNICATIONS

David B. Cooper
Division of Engineering
Brown University
Providence, RI 02912

Abstract

Our interest in this paper is in encoding pictures, within a certain class, in minimum average length binary sequences. The pictures of interest here are those arising in conferencing by telecommunications, and blackboard contents which must be communicated for purposes of televising lectures. The data compression problem is that of determining features which can be efficiently parametrized in hierarchical structures and adequately represent the graphic data. Useful features treated here include those which one might expect in dealing with line drawings but include more abstract ones as well. We estimate that data compressions of better than 100 to 1 are possible.

Introduction

Fundamentally, the problem of effective graphic data compression is that of deciding on an acceptable degree of distortion which can be tolerated in the reconstructed pictures and subject to this limitation encoding the pictures to be sent in binary sequences of shortest average length. This permits picture storage in memory of smallest size or facsimile communication over a channel of some capacity C in minimum time. In this paper, we deal with good encoding of certain types of pictures, namely, those which arise in conferencing and remote classrooms. The importance of low cost communication of this class of pictures for the purpose of reducing the need for human transportation or for providing services that have, heretofore, been too expensive is self-evident. In conferencing, one is usually dealing with drawings which can be made on a pad, with draftsman prepared charts, or with slide or viewgraph projections. In the remote classroom, one is interested in pictures drawn on the blackboard. In these cases, the pictures in question are almost always line drawings, and more specifically, line drawings belonging to a limited class. It is exploitation of the structure of this rather restricted class of two-dimensional data that permits the use of parametrizations which we estimate should result in data compressions greater than 100 to 1. It is our feeling that the approach described here will lead to essentially the greatest data compression reasonably possible for the class of pictures under consideration and will provide further insight into considerations important for encoding more complex classes of pictures. Figures 1 and 2 are pictures which have appeared in classroom lectures on system theory or pattern recognition. Note that for the most part, we are not dealing with arbitrary curves but rather those that comprise Roman or Greek letters, punctuation symbols, mathematical symbols, Arabic numbers, straight lines, piecewise-linear curves, rectangles, circles, ellipses, and others. In other words, we are dealing with a rather limited class of objects, each object constructed of lines and arcs, and it is usually the objects that must be communicated with high intelligibility rather than the component lines or arcs themselves. The emphasis in this paper is on dealing with parametrizable subpictures. However, as we briefly comment further on, aspects of the approach are also of importance when encoding to accurately reproduce the data. We also point out that the price of high compression is complex pattern recognition. However, extensive P.R. at the encoder can be economic because of the high cost of long distance communications.

Structure, Distortion, Statistical Independence, and Clustering

The usual conversion of a picture to digital representation is to perform a raster scan consisting of a sequence of line scans capable of resolving the picture field into an array of rectanglar resolution cells -- 2^K cells per line and $2^{K'}$ cells per column. The data raster than consists of an array of quantized gray levels, one of two levels for each resolution cell. Of course, the coarser the quantization, the more compact can be the encoded representation of the picture, but also the greater is the distortion. However, it has been recognized (see e.g., references 1 and 2) that by parametrizing simple subpictures such as lines, arcs, circles, etc., data can be encoded more compactly. Our paper is a further development of this idea. However, we pause to point out that if essential subpicture structure is preserved (e.g., 4 linear sides intersecting at right angles for a rectangle), then very coarse quantization of the subpicture parameters may be acceptable (e.g., poor reproduction of rectangle size, location and orientation may often be acceptable).

We briefly recall a few elementary results concerning data encoding. Let X be an n-component discrete random vector. We wish to encode each point in the range of X in a code word of 0's and 1's such that average code word length is a minimum. The lower bound on this length is the entropy H(X). It is possible to find a code which comes close to achieving this bound, and such a code assigns a binary sequence of length roughly $-\log p(x)$ to the event $X = x$, where $p(x)$ is the probability of the event and the base of the logarithm is 2. Furthermore, if the components of X are statistically independent, then $H(X) = \sum_k H(X_k)$ (where X_k is the kth component of X) and little increase in the minimum average number of binary digits required to encode X is incurred through encoding the components of X individually. In theory then, a picture can be considered to be an observation of a random vector and the minimum average number of binary digits necessary for encoding pictures of interest should be close to the associated entropy. Since this is clearly impossible to implement for several reasons, we consider an alternative of decomposing a picture into subpictures such that these subpictures are relatively independent of one another but each subpicture is composed of data which is highly dependent. The decomposition is repeated and these subpictures are themselves decomposed into subpictures, etc. Then in essence, picture representation is that of a hierarchy of abstract clusters. The elements of a cluster are abstract representations of subpictures and are described by symbols and spatial parameters. Each cluster is encoded within the cluster of which it

is an element. If the clusters have been well chosen, then each cluster is an element of a much smaller and simpler universe than that within which it would be characterized were it treated simply as a subpicture described by the coordinate system of the picture field. Consequently, simple and reasonably efficient codes can be used within the clusters at each level and this results in orders of magnitude of reduction in the number of binary digits used to represent a picture over that which is necessary when no attempt is made to remove redundancy.

On Simple Primitives Which Are Likely To Be Useful

Examination of Figures 1 and 2 and similar material leads one to believe that some or all of the following simple primitives will be among those of importance in near optimum coding. We briefly comment on minimal parameter sets for characterizing each of the primitives considered. These primitives are circles, line segments, rectangles, ellipses, triangles, and others. (Of course, simple primitives of this sort are often similarly parametrized for use in graphics terminals.)

Circles are characterized by three numbers. Line segments are characterized by four numbers; two numbers for each end point. Rectangles are characterized by five numbers; various descriptors will do but one set is p_1, p_2, & where p_1 is the lowest and then leftmost vertex, p_2 is the other end of that one of the two lines from p_1 which makes the smallest positive angle with the positive x-axis, and & is the length of the other side emanating from p_1. Ellipses are characterized by five numbers. Triangles are characterized by six numbers.

Encoding a primitive involves a binary sequence identifying primitive type and then a binary sequence encoding the parameter values. Obviously, the smaller the number of parameters, the shorter will be the associated code word.

A variety of other simple primitives can be considered as well. Among them are a larger class of quadratic curves including the parabolas and hyperboles, more generally q-ary curves, and other special functions such as damped sinusoids. The size of the set of primitives used depends on a number of factors which we touch on later.

On More Complex Primitives

Rectangles and triangles are examples of relatively simple primitives which can be realized as groupings of elementary primitives, namely, lines. Furthermore, most line and/or arc configurations can be well approximated by appropriate groupings of line segments. Consequently, is there an advantage in introducing a large number of primitives -- some of which will be quite complex? The size of an optimum primitive set depends on a number of factors. Among these are (i) the primitive set must obviously be capable of adequately representing the pictures under consideration. This adequacy should be measured by some distortion function. (ii) Along with the parameters identifying a primitive, there must be an identification of the type of primitive involved. The larger the set of primitives, the longer must be the binary sequences identifying primitive type. (iii) A complex primitive is usually (but not always) characterized by fewer parameters than is a representation of the primitive in terms of simpler primitives. As an example of a complex primitive, we introduce a connected straight line drawing. Figure 3 is an example of such a drawing. By "connected", we mean that any point on the drawing can be reached from any other point in the drawing by following an unbroken path within the drawing. There are a number of ways of parameterizing such drawings. One is to introduce nodes at all points of intersection. These are numbered arbitrarily in Figure 3. We then parametrize such a primitive by specifying a sequence of nodes such that successive nodes in the string are connected by straight lines in the figure. Two parameters are then needed for giving the coordinates of a node. A minimal length node string for Figure 3 is 1,2,3,9,8,7,6 5,2,4,13,12,10,11,10,4,5. There are 17 nodes in this sequence and 34 parameters provide a minimal description, of this type, for the figure. In general, a line drawing consisting of n connected line segments can be characterized by a number of parameters lying in the interval $[2(n+1), 2(2n-1)]$, and there are drawings for which the minimum number of descriptive parameters are the lower or upper bounds. Note the ratio of maximum length of optimal characterizing string to minimum length of optimal characterizing string for a connected n-line drawing is approximately 2 for $n \gg 1$. Hence, the connected-line drawing primitive provides a saving in the number of parameters required of at most 50% over encoding the drawing as a set of individual lines.

There is, of course, dependence among the members of the parameter sequence specifying node locations in this primitive. (Dependence may also exist among the spatial parameters characterizing the simpler primitives.) Much of this dependence can be removed by encoding differences in the x-coordinates and differences in the y-coordinates of successive nodes rather than encoding the coordinates of each node (a similarity to Freeman coding). This is also a simple way of reducing the size of the universe within which the parameters are encoded. Later in the paper we comment on the use of abstract clustering to facilitate the determination of more efficient codes.

Characters

By characters we refer to such objects as Arabic integers from 0 through 9, the upper and lower case letters of the Roman and Greek alphabets, English punctuation symbols such as period, comma, quotation marks, under bar, question mark, brackets, etc., and possibly others. In many cases, we are quite willing to have a word originally type in one font to be reconstructed at the receiving end as a word printed in another font and of a somewhat different size as long as the location is reasonably accurate (and reasonably accurate may mean very poor accuracy). If a standard font is used for characters with provision for arbitrary location, dilation, (possible other types of distortion such as rotation or stretching in one direction), then description of such a character requires identification of the character, a size parameter, and two location parameters.

Groups

The frequent occurrence of various complex picture types leads to the creation of new entities which we shall call "groups". Groups are simply ordered sequences of primitives, parameters, and other groups. Following are a few examples of groups of varying complexities.

Example 1. Suppose "is the power of 2x" appears as a line of letters. This sequence would be encoded as

the group type "symbol line". The only spatial parameters required for characterizing such a line are those specifying the location of the first symbol and the font size, a total of three parameters. Hence, the group is characterized by the sequence consisting of a symbol specifying group type, followed by a parameter specifying that there are 18 symbols in the sequence (includes four space symbols between words), followed by three spatial parameters, followed by the sequence of letters, numbers and space symbols in "is the power of 2x". Note that the major saving incurred through use of this group is that only three spatial parameters are required for providing spatial information for all letters and numbers in the group. A second saving incurred through use of this group is that the symbols are identified within a smaller universe of symbols. Only symbols such as English, and perhaps Greek letters, and Arabic numbers appear in these symbol sequences. Hence, these symbols are treated as members of a universe of 64 symbols rather than one of the order of 1000 which includes primitives, groups, and operators. A binary sequence of length 6 rather than of length 10 is then adequate for symbol representation. An increase cost of using group encoding is that group type identification and sequence length specification is now required.

Example 2. Suppose the capital letters B, E, and Y occur at various places in a picture but not in a meaningful group such as the "symbol line" type of Example 1. Then a meaningful group for these symbols is simply "English Capital Letters" group type. Suppose B occurs in two places and E and Y each once. Let X_{B1}, Y_{B1}, ℓ_{B1} and X_{B2}, Y_{B2}, ℓ_{B2} denote the spatial parameters representing symbol location and font size for each of the two B's in question and denote spatial parameters for E and Y similarly. Within the "English Capital Letters" group, B, E, and Y are put in sequence in some predetermined order such as the same order as they occur in the alphabet. Then the parameter sequence we use for this group type is the symbol for "English Capital Letters", followed by a parameter indicating that there are three next level subgroups, followed by the symbol for group B, followed by a parameter indicating two members in the "B" group, followed by X_{B1}, Y_{B1}, ℓ_{B1}, X_{B2}, Y_{B2}, ℓ_{B2}, then followed by the symbol for the "E" group, etc.

Savings can be realized through use of this group in two ways. First, if there are many occurrences of each of one or more letter types, then a single usage of the symbol for a letter type suffices for all occurrences of that letter. Second, additional code shortening structure can be built into the group. If the characters in a subgroup of letter type can be arranged in sequence such that distances between successive members of the sequence are 2^{-3} or less of the horizontal and vertical extents of the picture field, then parametrizing spatial information for the subgroup as the sequence of changes in location and size of one character to the next can result in a sequence of much smaller numbers than if the character spatial coordinates and size are encoded individually with respect to the coordinates for the picture field.

On Spatial Clustering

When encoding without exploiting subpicture structure, there are natural two-dimensional extensions of one dimensional coding techniques. A subpicture can be put into a small enclosing rectangle and the data points within coded with respect to a local coordinate system for the interior of the rectangle. Since the data points are then specified within a smaller universe of points, shorter binary sequences can be used for encoding data point locations. Note that we have introduced a new space dominated group here. If the saving in smaller number of binary digits for specifying data point locations is greater than the number of binary digits for specifying group type and the five parameters describing the enclosing rectangle, a saving can be realized. (Four spatial parameters will describe the enclosing rectangle if the sides are taken to be parallel to the X and Y coordinate axes.)

We comment briefly on the matter of when substantial saving can be realized by encoding within a restricted rectangular field. Suppose, e.g., that a line (say horizontal) with n points, or any other configuration of n points, is to be encoded. If the picture raster is M rows by M columns and $n \ll M^2$, then the following coding scheme is as good as any other. Individually encode the locations of the n data points. This requires 2n log M binary digits (log M^2 digits per point). If the data does constitute an essentially horizontal or vertical line, a smallest enclosing rectangle will have one side of length n and one side considerably shorter, so that many fewer than 2n log n bits will be required for the within-rectangle encoding.

Next, one considers whether subdividing the rectangle into an array of smaller rectangles is of use. Assume that the smallest square containing n points is an n by n square. Suppose the square is subdivided into an array of subsquares which are k by k and that m of these squares contain the n data points. Then roughly 2m log (n/k) bits are necessary for specifying the m subsquares which contain one or more data points, m log n bits are required for specifying the number of data points within each subsquare (when $m \ll n$), and 2n log k bits are required for locating the data points within the subsquares. Hence, a total of roughly 2m log (n/k) + m log n + 2n log k bits are required here. If, for example, the n data points constitute a horizontal line, then m = n/k, and 2(n/k) log (n/k) + (n/k) log n + 2n log k bits are required. For n = 512, the minimum value of this expression is 4,416 bits -- achieved for k = 8. The number of bits required for encoding within the original n by n rectangle is 2n log n = 9,216. Hence, the maximum saving here is only 50%. However, if the n points do not constitute a single straight line but rather a set of straight lines, curves, or more generally scattered points which are contained in fewer than m = n/k of the $(n/k)^2$ subrectangles, then impressive savings can result from subdivision. Furthermore, it may then be worthwhile to again subdivide a number of these smaller rectangles, and there is some hierarchy of subdivisions which is optimum. Note that this is simply the recognition and encoding of a hierarchy of spatial clusters.

Other Clustering and Other Uses of Clusters

The preceding clustering involves encoding within a hierarchy of Cartesian coordinate systems. Other coordinate systems are sometimes more useful. For example, if some data is highly concentrated near the points of an elliptic or circular curve, then a simple and efficient encoding may be realized by specifying data point location with respect to the elliptic or circular curve. The efficiency would result from the simpler specification of the local universe within which the data points would be encoded.

Finally, we mention that these clustering methods can also be used in encoding the spatial parameters appearing in many primitives and groups.

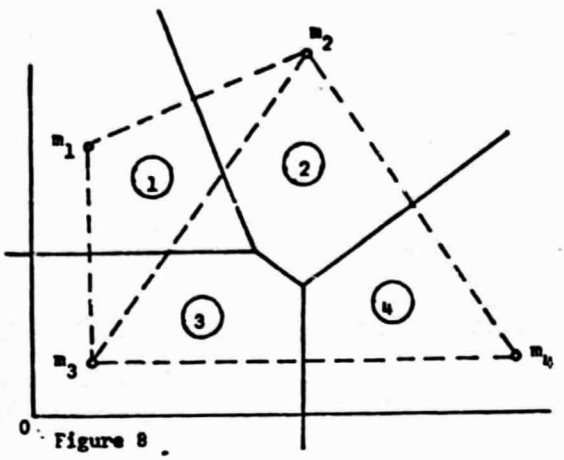


Figure 8

Figure 1

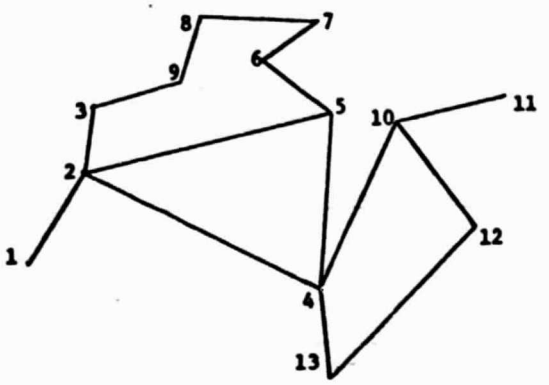
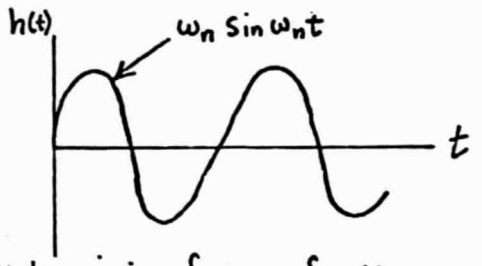


Figure 3

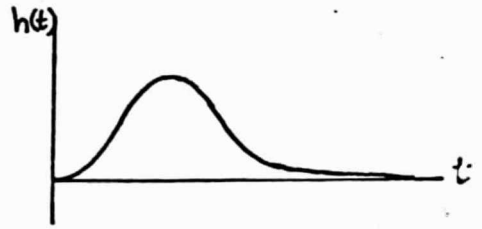
Limiting Cases

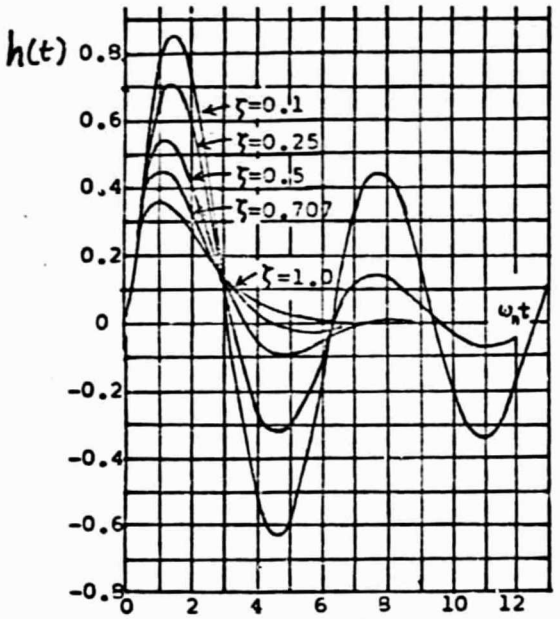
$\zeta = 0$ Undamped



Note origin of name for ω_n.

$\zeta = 1$ Critically Damped





Response to an impulse

Step Response $r(t) = \int_0^t h(\tau)\,d\tau$

$$r(t) = 1 - \frac{1}{\sqrt{1-\zeta^2}}\, e^{-\zeta\omega_n t} \sin\left(\omega_n\sqrt{1-\zeta^2}\,t + \cos^{-1}\zeta\right), \quad t>0$$

Figure 2

Examples

Following are rough estimates of the lengths of binary strings for encoding the pictures of figures 1 and 2 when using some of the simpler subpictures discussed. No attempt has been made in this estimation at reaping the benefits available by exploiting the fact that a parameter doesn't take all its possible values with equal probability. Nor have we used the fact that the different letters, numbers, math symbols, etc., do not all occur with the same probabilities. Because of this, our estimates of binary sequence length to represent pictures is high. Shorter sequences can be realized through use of variable-length code words.

Figure 1: We assume adequate reproducibility using a raster requires roughly five vertical and

five horizontal lines per letter (this number is probably too low). Then the figure is roughly 280 lines high and 360 line wide, or equivalently, the uncompressed data in a raster necessitates a binary string of 280 x 360 = 100,800 digits to represent the figure. Encoding using our methods would consist of the following. We shall encode all parameters to an accuracy of one part in 512, i.e., to a quantization error of less than one line as used in the standard scanning above. Hence, nine bits are needed to specify a parameter value. As mentioned previously in the proposal, such accuracy for our method is wholely unnecessary; a quantization error four or more times as large would be perfectly adequate. We assume a letter, number, punctuation, mathematical symbol, set of at most 128 symbols. Hence, each such symbol can be encoded in a binary sequence of length seven. The table below summarized the parameter and bit amounts required for characterizing the various portions of the figure. Since the 13 symbols appear in a group and all have the same font, the symbol size parameter must be used only once and therefore does not appear in the table.

Primitive or Group Type	Number of Parameters	Number of Bits
10 lines comprising 2 connected line drawings	8 nodes in one sequence, 5 nodes in other = 26 parameters	234
2 coordinate axes	6 parameters	54
4 circles in a group	12 parameters	108
13 symbols in a group - letters and numbers	2 params/symbol = 26 params	91 bits to identify symbols plus 234 bits for params = 325
1 symbol line of 8 letters and numbers	Starting point for line, number of symbols, and symbol size -- 4 params	56 bits to identify symbols plus 36 bits for params = 92
		TOTAL 813
		ADJUSTED TOTAL 1000

The total did not take into account bits required for identifying and characterizing primitives and groups. Consequently, we rounded the total of 813 bits to 1000. This number can probably be reduced further.

Figure 2: At roughly 5 lines per smallest character, this figure requires a raster of 672 x 768 lines or 516,096 bits. Hence, specifying parameters, in our coding system, using a parameter quantization interval of somewhat less than one line requires 10 bits (or a binary sequence of length 10) to specify a parameter value. We assume there are just one or two number and letter sizes, thus necessitating only a small number of bits for coding these parameters, and consequently, only two spatial parameters per symbol or symbol sequence are entered into the table. The cost of our encoding is then computed as follows:

Primitive or Group Type	Number of Parameters	Number of Bits
7 damped sinusoids	4 curve parameters plus start and stop points = 8 params/curve or 56 params	560
1 grid	7	70
2 pairs of coordinate axes	16	160
6 lines	24	240
6 lines with arrowheads	24	240
60 essentially individually appearing symbols including symbols in equations	2 spatial parameters/symbol = 120 params	420 bits to specify symbols plus 1,200 for spatial parameters = 1,620
6 lines of words, about 120 symbols; 21 two to four digit decimal numbers or parameter values in the graph and symbol sequences elsewhere -- about 95 symbols	2 parameters/line = 54 spatial params	1,505 symbol bits plus 540 spatial bits = 2,045
		TOTAL 4935
		ADJUSTED TOTAL 6000

The data raster involves 516,096 bits, so a coded picture of 6000 bits realizes a compression of about 86 to 1. Were we to use a more sophisticated encoder, the graph could be treated as a group, operators for superscripts and subscripts introduced, etc., and additional compression realized.

References

[1] Freeman, H., "A Review of Relevant Problems in the Processing of Line-Drawing Data", in Automatic Interpretation and Classification of Images, edited by A. Grasselli, Academic Press, pp. 155-174, 1969.

[2] Conopac Manual, "Conopac Subroutines for Conograph-12, version 2.6", Hughes Aircraft Co., 1973.